Basics

Libraries

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library unisim;
use unisim.vcomponents.all;

library work;
```

Basics

Signals, variables & constants

```
<prefix> <name> : <type> [:= <default>];

<prefix> can be one of the following:
* signal
* variable (only in processes)
* constant

<type> can be one of the following:
* std_logic
* std_logic_vector(<range>)
* signed(<range>)
* unsigned(<range>)
* integer
* integer range <lower> to <upper>
* boolean

<range> defines the size and direction of
the vector/array:
* std_logic_vector(0 to 7)
* (un)signed(7 downto 0)

<default> sets a default value:
* std_logic: single bit '0', '1', or 'Z'
* integer: integer value
* boolean: true or false
* std_logic_vector and (un)signed:
  - vector of bits "0Z1"
  - hex values x"ABCD"
  - aggregates (0 => '1', others => '0')
```

Package

Types & alias

```
subtype <name> is <type>;

type <name> is array(<range>) of <type>;

type <name> is (<value>, ...);

type <name> is record
    <name> : <type>;
    <name> : <type>;
end record;

alias <name> : <type> is <signal>;
```

Basics

Type conversion

```
integer to_integer(<(un)signed>)
signed to_signed(<integer>, <size>)
unsigned to_unsigned(<integer>, <size>)

vector std_logic_vector(<(un)signed>)
signed signed(<vector>)
unsigned unsigned(<vector>)
```

Basics

Operators

```
Logique: and/nand, or/nor, xor/xnor, not
Math: +, -, *, /, **, abs, mod, rem
Shift: sll, srl, sla, sra, rol, ror
Test: =, /=, <, >, <=, >=
Concatenation: &
Assignment: <=, :=
Selection: vector(0), vector(3 downto 2)
```

Basics

Entity

```
entity <name> is
generic(
   <constant> : <type> := <value>;
   <constant> : <type> := <value>
);
port(
   <signal> : <mode> <type>;
   <signal> : <mode> <type>;
);
end <name>;

<mode> can be:
* in
* out
* inout
```

Basics

Architecture

```
architecture <name> of <entity> is
   -- signals
begin
   -- statements
end <name>;
```

Basics

Entity instanciation

```
<name> : entity work.<entity>
generic map(
   <constant> => <value>,
   <constant> => <value>
)
port map(
   <signal> => <signal>,
   <signal> => <signal>
);
```

Sequential

Process

```
process(<signal>, ...)
   -- variables
begin
   if (rising_edge(<clk>)) then
      <statement>;
   end if;
end process;
```

Sequential

Case

```
case <signal> is
   when <choice> =>
      <statement>;
   when <choice> | <choice> =>
      <statement>;
   when <lower> to <upper> =>
      <statement>;
   when others =>
      <default>;
end case;
```

Sequential

If / elsif / else

```
if <condition> then
   <statement>;
elsif <condition> then
   <statement>;
else
   <default>;
end if;
```

Sequential

For

```
for <name> in <range> loop
   <statement>;
end loop;
```

Sequential

While

```
while <condition> loop
   <statement>;
end loop;
```

Combinatorial

With

```
with <signal> select <name> <=
   <value> when <choice> | <choice>,
   <value> when <lower> to <upper>,
   <default> when others;
```

Combinatorial

When

```
<name> <= <value> when <condition> else
          <value> when <condition> else
          <default>;
```

Combinatorial

Generate

```
<label>: if <condition> generate
begin
   <statement>;
end generate;

<label>: for <name> in <range> generate
begin
   <statement>;
end generate;
```

Package

Function

```
function <name>(
   signal <name> : <type>;
   ...
) return <type>;

function <name>(
   signal <name> : <type>;
   ...
) return <type> is
   -- variables
begin
   <statement>;
end <name>;

<signal> <= <function>(
   <input1> => <insig1>,
   <input2> => <insig2>,
   ...
);
```

Package

Procedure

```
procedure <name>(
   <prefix> <name> : <mode> <type>;
   ...
);

procedure <name>(
   <prefix> <name> : <mode> <type>;
   ...
) is
   -- variables
begin
   <statement>;
end <name>;

<procedure>(
   <input1> => <insig1>,
   ... ,
   <output1> => <outsig1>,
   ...
);
```

Advanced

Xilinx Primitives

```
-- Input clock buffer
IBUFG_inst : IBUFG
generic map(
   IBUF_LOW_PWR => TRUE | FALSE,
   IOSTANDARD => “DEFAULT”
)
port map (
   O => O, -- buffered clock
   I => I -- input pin
);

-- Clock buffer
BUFG_inst : BUFG
port map(
   O => O, -- buffered clock
   I => I -- input signal
);

-- Clock muxer
BUFGMUX_inst : BUFGMUX
generic map(
   CLK_SEL_TYPE => “SYNC” | “ASYNC”
)
port map(
   O => O, -- output clock
   I0 => I0, -- input clock 0
   I1 => I1, -- input clock 1
   S => S, -- source select
);

-- Input buffer
-- see OBUF for output buffer
IBUF_inst : IBUF
generic map(
   IBUF_LOW_PWR => TRUE | FALSE,
   IOSTANDARD => “DEFAULT”
)
port map(
   O => O, -- buffered signal
   I => I -- input signal
);

-- Differential input buffer
-- see OBUFDS for output buffer
IBUFDS_inst : IBUFDS
generic map(
   DIFF_TERM => FALSE | TRUE,
   IBUF_LOW_PWR => TRUE | FALSE,
   IOSTANDARD => “DEFAULT”
)
port map(
   O => O, -- buffered signal
   I => I, -- input positive
   IB => IB -- input negative
);
```

```
-- Bi-directioal buffer
IOBUF_inst : IOBUF
generic map(
   DRIVE => 12,
   IOSTANDARD => “DEFAULT”,
   SLEW => “SLOW”
)
port map (
   O => O, -- from the world
   IO => IO, -- inout pin
   I => I, -- to the world
   T => T -- select: high=in, low=out
);
```

Advanced

Attributes

X’high	upper bound of X
X’low	lower bound of X
X’left	leftmost bound of X
X’right	rightmost bound of X
X’range	range of X
X’length	length of X
X’event	true of fired the process
X’last_value	previous assigned value

Advanced

UCF

```
NET “<port>” <attributes> [| ...];

<attributes> can be one or more:
* LOC = <pin>
* IOSTANDARD = <io_standard_name>
* IOBDELAY = <NONE/BOTH/IBUF/IFD>
* DIFF_TERM = <TRUE/FALSE>
* SLEW = <SLOW/FAST>
* DRIVE = <2/4/6/8/12/16/24>
```

Advanced

Simulations

```
wait for <time>;
wait;

report “<string>”;

assert <condition> [report “<string>”]
[severity ERROR|WARNING];
```

Tips

Best Practice

```
* Only use IN and OUT modes
* Give _i or _o suffixes to signals
* Only use STD_LOGIC or STD_LOGIC_VECTOR
* Vectors always start at 0
* Vectors always go from MSB downto LSB
* Register all the outputs

* Use synchronous resets
* Initialize all signals in reset
* All signals are active high
* Use the rising edge of the clock
* Synchronous processes only have the
  clock as sensitive signal

* The file’s name is the entity’s name
* Use testbenches
* Use named signal mapping for entities
* Cover all the cases in conditions
* Use constants when possible
* Avoid variables inside processes
```